|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 刘臣 | **学号** | 17 |
| **实验题目** | Lab2 | | |
| **实验内容** | 1.（练习1）了解ucore的“项目组成”。  使用ls查看ucore中lab1的项目组成    2.（练习1）会使用make编译文件，了解Makefile中的主要组成，实验报告中要有简单介绍。查看使用make与make “V=”所生成信息的不同。 根据make编译信息，简要说明gcc编译器是如何一步步生成ucore可执行文件的。sign.c的作用是什么。  打开lab1，查看Makefile文件，在Makefile文件中，生成ucore.img的相关代码为  | $(UCOREIMG): $(kernel) $(bootblock)  | $(V)dd if=/dev/zero of=$@ count=10000  | $(V)dd if=$(bootblock) of=$@ conv=notrunc  | $(V)dd if=$(kernel) of=$@ seek=1 conv=notrunc  为了生成ucore.img，首先需要生成bootblock、kernel  使用make指令，命令行很简洁：    当使用make V=时，会看到更加详细的操作过程，最后生成.o文件    编译器gcc预处理，生成预编译文件（.i文件）：gcc –E main.c –o main.i  编译，生成汇编代码（.s文件）：gcc –S main.i –o main.s  汇编，生成目标文件（.o文件）：gcc –c main.s –o main.o  链接，生成可执行文件（executable文件）：gcc main.o –o main  Sign工具是处理bootblock.out，生成bootblock  3.（练习2）学会使用qemu与gdb协作进行调试ucore代码；从CPU加电后执行的第一条指令开始，单步跟踪BIOS的执行。在初始化位置0x7c00设置实地址断点,测试断点正常。从0x7c00开始跟踪代码运行,将单步跟踪反汇编得到的代码与bootasm.S和 bootblock.asm进行比较。自己找一个bootloader或内核中的代码位置，设置断点并进行测试。（截图并简要说明实验过程）  说明：中间实验的时候因为原来的Linux系统没有gnome，而我对于自带的terminal编程不熟悉，所以换了个Ubuntu16的系统  修改tools里面的gdbinit和Makefile里面的debug部分，然后执行make debug    查看BIOS的代码    再次修改gdbinit，在初始化位置0x7c00设置实地址断点,测试断点正常    下面是q.log和bootasm.S和 bootblock.asm中的代码对比    4.（练习3）了解如何在bootloader中完成由实模式到保护模式的转换的。（要有过程或对应代码的截图）      5.（练习4）根据代码，通过阅读bootmain.c，了解bootloader如何加载ELF文件。（需要截图，并简要说明）        6.（练习5）要求完成函数kern/debug/kdebug.c::print\_stackframe的实现.完成kdebug.c中函数print\_stackframe的实现。（需要make qemu 后的结果等截图，简要说明打印信息，说明调用关系。）    ss:ebp指向的堆栈位置储存着caller的ebp，以此为线索可以得到所有使用堆栈的函数ebp。  ss:ebp+4指向caller调用时的eip，ss:ebp+8等是（可能的）参数。  输出中，堆栈最深一层为  ```  ebp:0x00007bf8 eip:0x00007d68 \  args:0x00000000 0x00000000 0x00000000 0x00007c4f  <unknow>: -- 0x00007d67 --  ```  其对应的是第一个使用堆栈的函数，bootmain.c中的bootmain。  bootloader设置的堆栈从0x7c00开始，使用"call bootmain"转入bootmain函数。  call指令压栈，所以bootmain中ebp为0x7bf8。  7.（练习6）从代码找出中断描述符表（也可简称为保护模式下的中断向量表）的定义，并简要说明中断描述符表中一个表项占多少字节？各分别表示什么？其中哪几位代表中断处理代码的入口？（截图并简要说明）  中断向量表一个表项占用8字节，其中2-3字节是段选择子，0-1字节和6-7字节拼成位移，两者联合便是中断处理程序的入口地址。    8.（练习6）请编程完善kern/trap/trap.c中对中断向量表进行初始化的函数idt\_init。在idt\_init函数中，依次对所有中断入口进行初始化。使用mmu.h中的SETGATE宏，填充idt数组内容。每个中断的入口由tools/vectors.c生成，使用trap.c中声明的vectors数组即可。（截图并简要说明）    9.（练习6）请编程完善trap.c中的中断处理函数trap，在对时钟中断进行处理的部分填写trap函数中处理时钟中断的部分，使操作系统每遇到100次时钟中断后，调用print\_ticks子程序，向屏幕上打印一行文字”100 ticks”。 （截图并简要说明）    10.参考答案labcodes\_answer/lab1,并在labcodes/lab1中完成challenge1内容，并简要说明实现的过程（设计到哪些函数，分别在哪个文件中，执行中断的过程中，先后使用了哪些函数。）。 | | |
| **总结** | 1. ucore项目组织形式和调试方法 2. ucore启动流程 3. ucore中断机制 4. 内核态与用户态的切换流程 | | |
| **日期** | 2020.05.31 | **成绩** |  |